# IP RX and IP TX Overview

Team Gamma – Antony Gillette

April 16, 2017

## Overview

- The IP RX and IP TX modules can be found inside the integration and testbench zips but can also be seen online at the following two links:
    - https://github.com/gamma-team/ip-rx-hdl/blob/master/src/ip_rx.vhd
    - https://github.com/gamma-team/ip-tx-hdl/blob/master/src/ip_tx.vhd
- The following is an overview of the functionality

## IP RX

- The input data is an IPv4 header followed by a UDP datagram
- The output data in order is Protocol, Source IP, Dest IP, UDP datagram
- The first 8 stages of the 9 stage pipeline are for the 8 bytes of input
    - Each stage has a case statement using a read length counter
    - The read length counter at the start is incremented by the number of valid bytes seen, from there it's incremented by one for valid bytes
    - Data is streamed directly from the input to output, so valid bytes are set to 0 when data is received that isn't needed by UDP RX
        - This is why the protocol byte position is first followed by source IP and destination IP for the output and UDP RX input
    - For case 9 (protocol) and cases 12 and after (source/destination IP followed by UDP datagram), the output valid bit is set to '1'
    - For each byte read in the header, it is added to the accumulated checksum and shifted left 8 bits first if it would correspond to the upper 8 bits when breaking up the header into 16 bit segments
- The 9th stage of the pipeline handles output while calculating the checksum
    - The checksum overflow needs to be added in before being compared to x"FFFF" and if it doesn't match then that means the checksum is incorrect

- The IP RX module was rewritten once while the IP TX module was written twice so there’s a noticeable difference in quality between the two latest versions and even more so compared to the older versions

## IP TX

- The input data in order is Source IP, Destination IP, Protocol, UDP datagram
- The output data is an IPv4 header followed by a UDP datagram
- The first 8 stages of the 9 stage pipeline are for the 8 bytes of input
    - Each stage has a case statement using a read length counter
    - The read length counter at the start is incremented by the number of valid bytes seen, from there it’s incremented by one for valid bytes
    - The first 9 cases are to store the IP source and destination addresses and cases 13 and 14 read the data length in the UDP header and adds 20 (the additional length of the IP header)
    - After the first 9 cases, data is stored into a data buffer and a corresponding data valid byte is set
    - In the 8th stage of the pipeline, when the packet length is read, the 9th stage is enabled and the calculation for the checksum is started
- The 9th stage of the pipeline handles output while calculating the checksum
    - The first section of the 9th stage outputs the first 8 bytes of the IP header (including the calculated packet length) and does the first overflow calculation for the checksum
    - The second section of the 9th stage outputs the second 8 bytes of the IP Header including the checksum which has the second overflow calculation
        - After the first overflow, for the summation of 20 bytes the maximum the overflow could be is one extra bit which is how the second overflow and output can be done in one stage with signals instead of variables
    - The third section of the 9th stage outputs the last 4 bytes of the IP header and the first 4 bytes of the buffer if filled
        - If the buffer isn’t filled yet but the end signal isn’t seen, it accordingly sets the output buffer place and goes to the next section

        - If the buffer isn't filled and the end signal is reached, it sets the data end bit, skips the next section, and goes straight to reset
    - The fourth section of the 9th stage outputs data in the buffer, and if the output outpaces the placing of data in the buffer, then it waits for the data to be read by keeping the pointer at the last read position while clearing the bytes successfully read
        - When the buffer is empty and the end signal is seen, it goes on to the next section
    - The fifth section resets the necessary signals for the next packet
- A 128 byte buffer is used because a 64 byte buffer results in conflicts and data corruption
    - 64 bytes should be enough for the 1st and 8th stages to output to the buffer at the same time, so it could be a tool issue
    - This buffer uses up a lot of resources so a 72 byte buffer could be enough to fix the issue
- The error signal doesn't have as much importance for TX as it does RX because the only thing that needs to be checked is if the UDP protocol byte is set properly, as both checksum and packet length are calculated instead of read